Lecture 14

Multimedia Data and Its Encoding

M. Adnan Quaium

Assistant Professor
Department of Electrical and Electronic Engineering
Ahsanullah University of Science and Technology
Room – 4A07
Email – adnan.eee@aust.edu
URL- http://adnan.quaium.com/aust/cse4295

Text Encoding

The character by character encoding of an alphabet is called encryption. As a rule, this coding must be reversible – the reverse encoding known as decoding or decryption.

Examples for a simple encryption are the international phonetic alphabet or Braille script

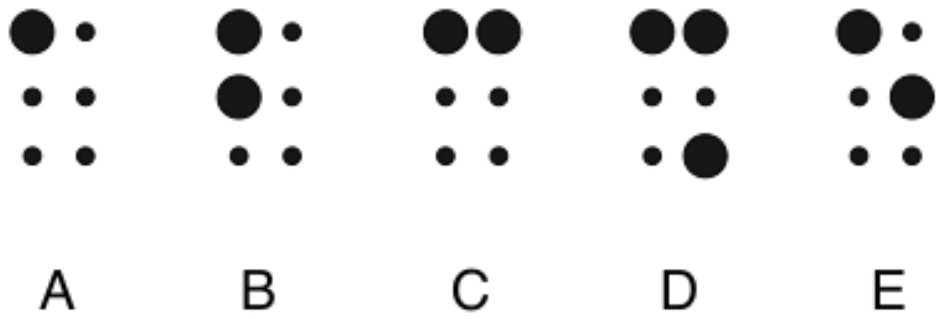
Table 4.1 The International Phonetic Alphabet

Alpha	Bravo	Charlie	Delta	Echo
Foxtrott	Golf	Hotel	India	Juliette
Kilo	Lima	Mika	November	Oscar
Papa	Quebec	Romeo	Sierra	Tango
Uniform	Victor	Whiskey	X-Ray	Yankee
Zulu				

Text Encoding

Louis Braille and the Braille Writing System

Louis Braille (1809 – 1852) lost his eyesight in an accident as a child. He refused to to accept that his only access to literature would be when it was read to him and sought early in life to develop a form of writing that would make it possible for the blind to read and write.



In 1821, he published his easy to learn writing system called Braille. It had been developed from a complex "night writing" system designed for the military by artillery captain *Charles Barbier* (1767 – 1841). Commissioned by Napoleon, Barbier had invented night writing to make it possible for soldiers to communicate with each other without sound or light. However, because of its overly complicated system of dots and syllables the system writing turned out to be unsuitable for military use. Louis Braille simplified this writing by replacing the syllables with letters and reducing the number of dots per symbol from twelve to six. A letter could be easily felt with the tip of the finger, making it unnecessary to move the finger, which made rapid reading possible. Each letter of the writing system developed by Braille consists of six dots arranged in a 3x2 matrix. The encoding of the letters was done by raising certain points in the matrix so they could be felt with the fingertip.

Morse Code

Samuel F. B. Morse and Alfred Vail used a form of binary encoding, i.e., all text characters were encoded in a series of two basic characters.

The two basic characters – a *dot* and a *dash* – were short and long raised impressions marked on a running paper tape.

Word borders are indicated by breaks.

A	B	C	D	E	F	G	H	I	J	K	L	M
.-	-...	-.-.	-..	.	..-.	--.	....	..	.---	-.-	.-..	--
N	**O**	**P**	**Q**	**R**	**S**	**T**	**U**	**V**	**W**	**X**	**Y**	**Z**
-.	---	.--.	--.-	.-.	...	-	..-	...-	.--	-..-	-.--	--..

Morse Code

- To achieve the most efficient encoding of the transmitted text messages, Morse and Vail implemented their observation that *specific letters came up more frequently in the (English) language than others*.
- The obvious conclusion was to select a shorter encoding for frequently used characters and a longer one for letters that are used seldom.

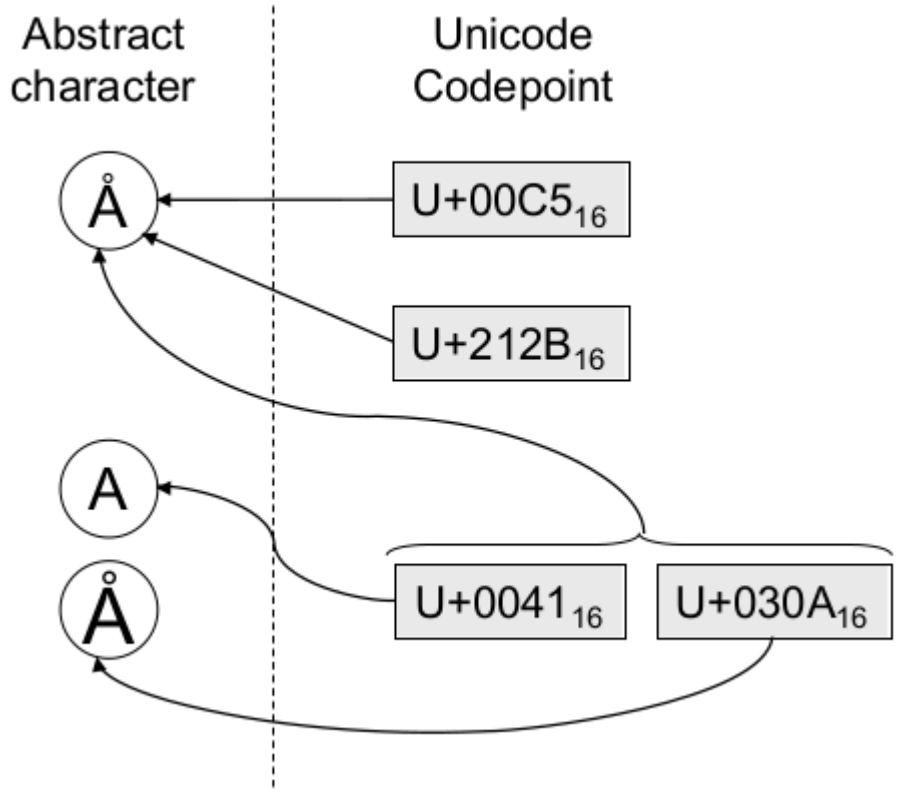
A	B	C	D	E	F	G	H	I	J	K	L	M
.-	-...	-.-.	-..	.	..-.	--.	....	..	.---	-.-	.-..	--
N	O	P	Q	R	S	T	U	V	W	X	Y	Z
-.	---	.--.	--.-	.-.	...	-	..-	...-	.--	-..-	-.--	--..

7 bit ASCII code

Binary	000	001	010	011	100	101	110	111
0000				0	@	P	'	p
0001			!	1	A	Q	a	q
0010			"	2	B	R	b	r
0011			#	3	C	S	c	s
0100			$	4	D	T	d	t
0101			%	5	E	U	e	u
0110			&	6	F	V	f	v
0111				7	G	W	g	w
1000			(	8	H	X	h	x
1001			)	9	I	Y	i	y
1010			*	:	J	Z	j	z
1011			+	;	K	[	k	{
1100			,	<	L	\	l	|
1101			-	=	M	]	m	}
1110			.	>	N		n	~
1111			/	?	O	_	o	

Unicode Standard

- The Unicode standard assigns a number (code point) and a name to each character, instead of the usual glyph.
- It represents each individual character in an abstract manner, while the visual representation of the character is left up to the text displaying software, e.g., the web browser.
- Characters can be assigned to several different code points since the same characters often belongs to different writing systems.
- Some characters are assigned several code points.
- It is also possible for one character to be composed of several basic characters that exist separately.



Unicode Encoding

- In Unicode encoding, the *first 256 characters* are assigned characters of the ASCII code to ensure a compatibility between the old ASCII encoding and Unicode.
- Unicode characters are usually displayed in the form U+xxxxxxxx , whereby xxxxxxxx stands for a code point in hexadecimal format. Leading zeros can be omitted.
- The code space provided in Unicode is divided into individual planes. Each of them contains 2^{16} =65,536 code points. Of these planes, currently 17 are available for use (as a result the character space, encodable by means of Unicode, is limited to 17×2^{16} =1,114,112 characters), with only planes 0–2 and 14–16 in use.

Plane	Title	from	to
0	Basic Multilingual Plane (BMP)	$U+0000_{16}$	$U+FFFF_{16}$
1	Supplementary Multilingual Plane (SMP)	$U+10000_{16}$	$U+1FFFF_{16}$
2	Supplementary Ideographic Plane (SIP)	$U+20000_{16}$	$U+2FFFF_{16}$
14	Supplementary Special-purpose Plane (SSP)	$U+E0000_{16}$	$U+EFFFF_{16}$
15	Supplementary Private Use Area-A	$U+F0000_{16}$	$U+FFFFF_{16}$
16	Supplementary Private Use Area-B	$U+100000_{16}$	$U+10FFFF_{16}$

Unicode Encoding

- The first level – plane 0 with the code points 0 – 65.535 – is called the *Basic Multilingual Plane (BMP)* and includes almost all spoken languages.
- The second level (plane 1), the *Supplementary Multilingual Plane (SMP)*, contains rarely used and mostly historical writing systems, such as the writing system of the Old Italian language, or a precursor to Greek, the Cretan writing systems.
- The third level or plane (plane 2), called the *Supplementary Ideographic PLane (SIP)*, contains additional, rarely used, ideographic characters from the “CJK“ group of Chinese, Japanese and Korean characters that are not classified in the BMP.
- Level 14, the *Supplementary Special-purpose Plane (SSP)*, contains additional command and control characters that were not assigned on the BMP. Planes 15 and 16 accommodate privately used characters.

Unicode UTF Transformation

The various UTF Transformations (UTF-7, UTF-8, UTF-16 or UTF-32) were developed to enable the efficient encoding of Unicode code points.

- UTF-8 is the most well-known variety and implements an encoding of variable length from 1–4 bytes.
- The first 128 bits of the Unicode, encompassing the characters of the 7-bit ASCII code, are only represented by one byte.
- The byte order of every UTF-8 encoded character starts with a preamble that encodes the length of the byte order.
- To ensure maximum compatibility with ASCII encoding, the 128 characters of the 7-bit ASCII code are assigned the preamble *0*.
  - If a UTF-8 encoded character consists of several bytes, the start byte always begins with a *1* and every succeeding byte with the preamble *10*.
  - With multi-byte characters, the quantity of of 1-bits in the preamble of the start bytes, gives the byte length of the entire UTF-8 encoded character.

Unicode UTF Transformation

1 byte	0xxxxxxx	(7 bit)
2 bytes	110xxxxx 10xxxxxx	(11 bit)
3 bytes	1110xxxx 10xxxxxx 10xxxxxx	(16 bit)
4 bytes	1111xxxx 10xxxxxx 10xxxxxx 10xxxxxx	(21 bit)

The shortest possible encoding variation is chosen for the UTF8-encoding of a code point. The Unicode code point is always entered in the encoding scheme right-justified. The following examples illustrate the principle of UTF-8 encoding:

Character	Codepoint	Unicode binary	UTF-8
y	$U+0079_{16}$	00000000 01111001	**0**1111001
ä	$U+00E4_{16}$	00000000 11100100	**110**00011 **10**100100
€	$U+20AC_{16}$	00100000 10101100	**1110**0010 **10**000010 **10**101100

For all scripts based on the Latin alphabet, UTF-8 is the most space-saving method for the mapping of Unicode characters.

Text Compression

Compression (compaction) techniques are methods to minimize the redundancy contained in a message and to use the available bandwidth as efficiently as possible.

- For text files 20% to 50% are a typical values for space saving, while savings of 50% up to 90% can be achieved with graphics files.
- However, with file types consisting largely of random bit patterns little can be achieved with these methods of compression.

Text Compression

Logical vs. physical compression

Semantic, or *logical*, compression, is achieved by continuous substitution, i.e., replacing one alphanumeric or binary symbol with another.

For example, the term *United States of America* is replaced with *USA*.

Syntactic or *physical* compression can be implemented on the given data without the information contained in the data being used.

For example,

- It was John who wore his best suit to the dinner last night.
- John wore his best suit to the dinner.

Text Compression

Symmetric vs. asymmetric compression

In *symmetrical compression*, encoding algorithms and decoding algorithms have approximately the same computational complexity. It's a different case with *asymmetric compression* methods.

Text Compression

Adaptive vs. semi-adaptive vs. non-adaptive compression

- Many methods (such as Huffman encoding) are used exclusively for the compression of certain media formats and thus use format-specific information. *This information is maintained in so-called dictionaries*.
- **Non-adaptive methods** use a static dictionary with predetermined patterns known to occur frequently in the information to be compressed.
  - Thus, a non-adaptive compression procedure for the English language could contain a dictionary with predefined character chains for the words “and, or, the“, because these words occur frequently in English.
- **Adaptive compression methods**, (LZW method) build their own dictionaries, with its own commonly found patterns for each application. These dictionaries are not based on predefined, application-specific patterns.
- **Semi-adaptive compression methods** present a mixture of both forms.

Text Compression

Lossless vs. lossy compression

- In *lossless* procedures the encoding and decoding of the data to be compressed is performed in such a way that the original data remains unchanged after completion of the process. *The information in the compressed data is fully preserved*.
  - Lossless compression methods are essential for text and program files.
- *Lossy compression* methods, attempt to achieve a higher compression rate by sacrificing parts of the information to be compressed, considered less important for the use intended.
  - For example, in audio compression lossy procedures dispense with sounds and sound sequences that cannot be perceived by the human ear.

Run Length Encoding (RLE)

The simplest type of redundancy in a text file are long sequences of repeated characters.

For example, the simple character sequence:

AAAADEBBHHHHHCAAABCCCC

This character string can be encoded in a more compact way:

4ADEBB5HC3AB4C

- Run length encoding is not feasible for a single character or for two identical letters because at least two characters are necessary for encoding.
- Very high compression rates can be achieved when long sequences of the same character occur.

Variable Length Encoding

Characters that often occur in the text often are assigned shorter code words than characters that only appear rarely.

For example, let us assume that the character sequence:

ABRAKADABRA

is to be encoded with a standard encoding that uses a 5-bit code for every letter of the alphabet

00001 00010 10010 00001 01101 00001 00100 00001 00010 10010 00001

Variable Length Encoding

Space saving can be achieved *if frequently used letters are encrypted with fewer bits* so as to minimize the total number of bits used for the string.

- The specified string sequence can be encrypted in the following way.
- First, the letters to be encoded are arranged according to the frequency of their occurrence.
- The top two letters are encoded with a bit sequence of the length 1.
- The following letters are encoded with for as long as possible with 2 bits each, subsequently with 3 bits each and so on.
- A can be encoded with 0, B with 1, R with 01, K with 10 and D with 11:

ABRAKADABRA

0 1 01 0 10 0 11 0 1 01 0

Variable Length Encoding

With this encoding, the addition of limiters (the spaces in the example) is also necessary between the separate letters to be encoded. If this is not done, *ambiguous interpretations* of the code words are possible.

Ambiguities can be avoided if attention is paid that no code starts with the bit sequence of another code.

Such codes are also referred to as Prefix Codes.

For example, we can encrypt A with 11, B with 00, R with 10, K with 010 and D with 011.

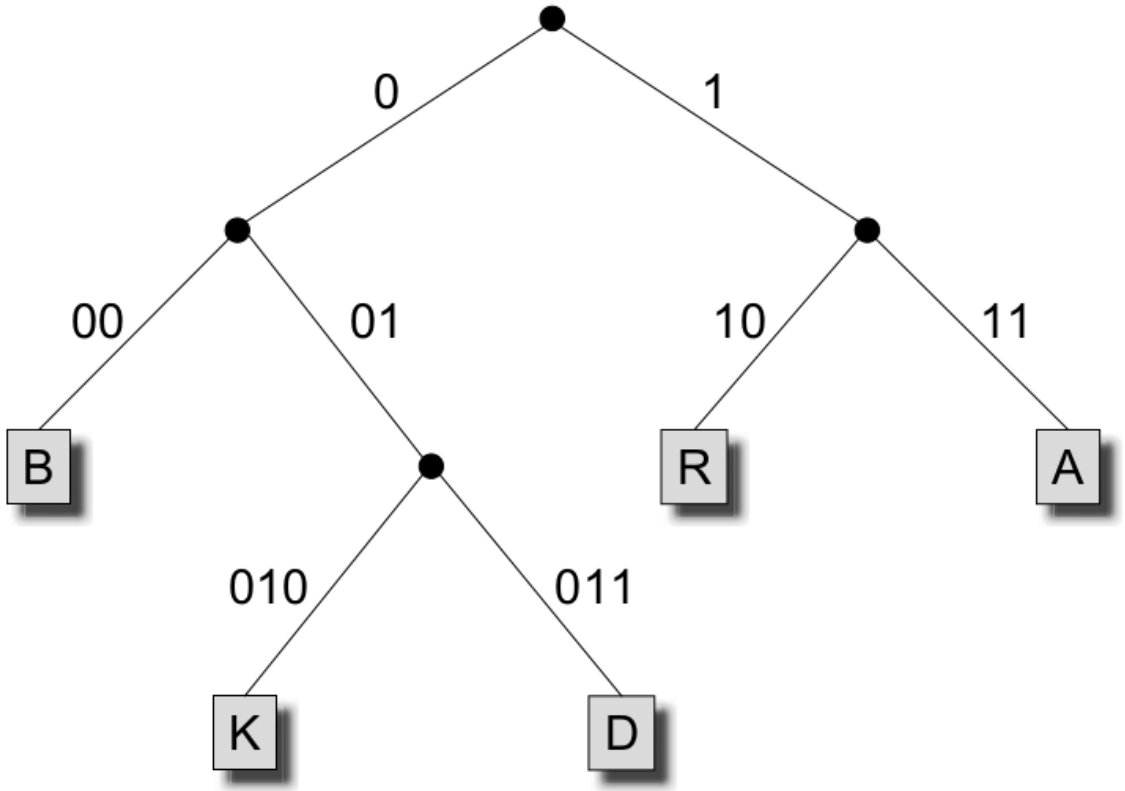
ABRAKADABRA
11001011010110111001011

The original 55-bit comprehensive standard coding could therefore be reduced to only 24 bits by encoding with variable length encoding.

Prefix Coding

The optimal encoding for a text file can be represented by a binary tree whose inner nodes always have two successors.

If the amount of N represents all letters to be encoded, then as an optimal prefix code for A the tree has exactly |N| leaf nodes and |N|-1 inner nodes.



Prefix Coding

If we consider a tree T, which corresponds to a predetermined prefix code, then the number of bits for the encoding of a predetermined file can be easily calculated.

- **f(c)** denotes the frequency with which a character c of the given alphabet A is found in our file.
- **$d_T(c)$** denotes the depth of the leaf node for the character c in the binary tree T, which incidentally corresponds to the length of the code word for c.

The number of necessary n bits B(T) to encode a file result in

$$B(T) = \sum_{c \in A} f(c) d_T(c)$$

Huffman Coding

The procedure developed by Huffman for the construction of an optimal prefix code works in the so-called bottom-up manner, i.e. it begins from below with a number of $|A|$ (unrelated) leaf nodes and leads to a a range of $|A|$-1 merger operations to construct a result tree. Besides bearing the letters $c \in A$, the leaf nodes also represent its frequency $f(c)$ within the encoded file. Next, the two nodes c_1 and c_2, which contain the smallest frequency data are selected. A new node c_{neu} is generated, which is marked with the sum from the two frequencies $f(c_{neu}) = f(c_1) + f(c_2)$ and connected to the two nodes selected as its successor. The nodes c_1 and c_2 are taken out of the amount A, while the new node c_{neu} is added to it. By proceeding in the same way, increasingly large subtrees are generated and number of the nodes in A becomes smaller and smaller. At the end all of the nodes are connected into one single tree. Nodes with a lower frequency are then the farthest away from the root node, i.e., they are also allocated the longest used code word, while codes with greater frequency are near the root nodes and have accordingly short code words. A tree generated in this way is a direct result of the Huffman code (see Fig. 4.7).

With the help of induction it can be shown that in fact the Huffmann method generates an optimal prefix code.

Huffman Coding

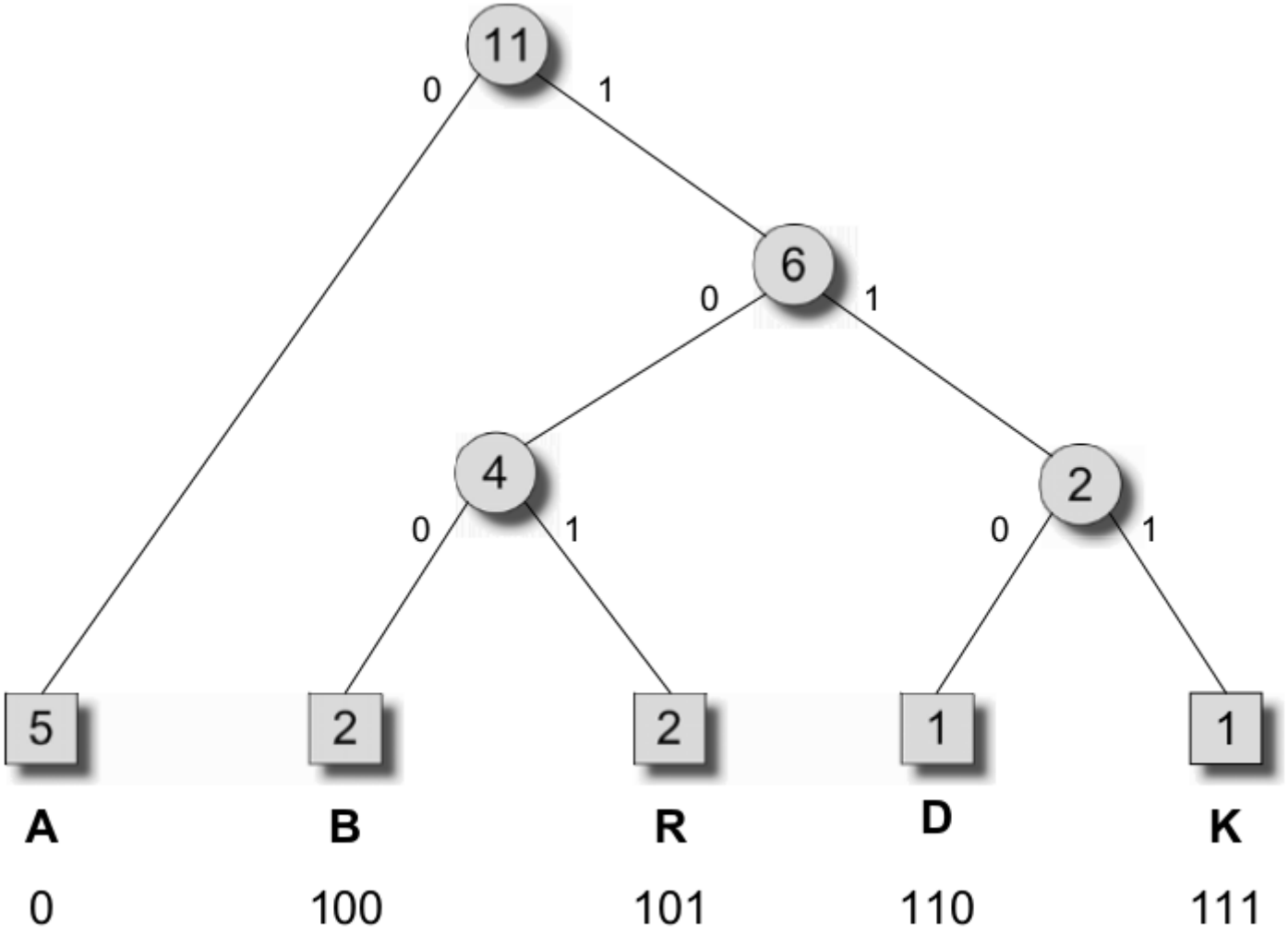


Fig. 4.7 Huffman coding represented by a binary tree.